# Module 3 Comprehensive Guide

## Keras and Deep Learning Libraries

## 📌 Deep Learning Libraries

Before building deep learning models, it is important to understand the **tools and libraries** that support their development. Over the past decade, several deep learning frameworks have emerged, each with its own strengths and design philosophy. This section introduces the **three most commonly used libraries** in practice and in this specialization: **TensorFlow, PyTorch, and Keras**.

### 🔹 Overview of Major Libraries

🛠 **TensorFlow**

1. **Developer:** Google
2. **Released:** 2015
3. **Usage:**
   * Most widely used deep learning library in both **research and production**.
   * Backed by a **large developer community**.
   * Actively maintained and improved with regular updates and enhancements.
4. **Strengths:**
   * Scalable for large models and systems.
   * Supports **deployment at scale**, including on mobile and edge devices.
   * Integrated with tools for production workflows (e.g., TensorFlow Serving, TensorFlow Lite).
5. **Limitations:**
   * Known for having a **steep learning curve**, especially for beginners.
   * Requires explicit graph construction and management of session contexts in its low-level APIs.

🛠 **PyTorch**

1. **Developer:** Meta(Facebook)
2. **Released:** 2016
3. **Origin:** Based on **Torch** framework written in Lua.
4. **Usage:**

* Rapidly adopted in the **academic research community**.
* Known for its dynamic computational graph and **Pythonic design**.
* Often preferred when working with **custom architectures** and experimental workflows.

1. **Strengths:**
   * **Intuitive and flexible** interface that feels native to Python users.
   * Excellent GPU support.
   * Increasing adoption in production through frameworks like TorchServe and ONNX.
2. **Limitations:**
   * Like TensorFlow, it also has a **learning curve**, particularly for complete beginners.
   * Lower-level model building can require more manual control compared to high-level APIs.

🛠 **Keras**

1. **Type:** High-level API
2. **Integration:** Runs on top of **TensorFlow** as backend.
3. **Usage:**

* Designed for **ease of use**, **clean syntax**, and **rapid prototyping**.
* Allows building **complex deep learning networks** with just a few lines of code.

1. **Strengths:**
   * Great for **beginners**, teaching, and small- to medium-scale projects.
   * Abstracts away many low-level details, allowing users to focus on the model structure and data.
   * Actively supported by Google.
2. **Limitations:**
   * Less fine-grained control compared to TensorFlow and PyTorch.
   * Advanced users may prefer to work directly with lower-level frameworks when custom behavior is required.

### 🔹 Summary of Comparison

| **Feature** | **TensorFlow** | **PyTorch** | **Keras** |
| --- | --- | --- | --- |
| Developer | Google | Meta (Facebook) | Initially François Chollet (Google) |
| Released | 2015 | 2016 | High-level API |
| Popularity | High in production and research | Increasing in research and industry | Very popular among beginners |
| Learning Curve | Steep | Moderate to steep | Easy |
| Ideal Use Case | Large-scale systems | Research, prototyping | Fast prototyping, learning |
| Control/Customization | High | High | Moderate |
| Backend Dependency | Native | Native | Runs on TensorFlow |

### 🔹 Takeaways

✅ The most widely used deep learning libraries today are **TensorFlow**, **PyTorch**, and **Keras**.

✅ **TensorFlow** is widely used in production, has robust community support, and is backed by Google.

✅ **PyTorch** is preferred in **academic research and custom model experimentation**, offering a flexible and Pythonic workflow.

✅ **Keras** is a high-level, beginner-friendly API that **simplifies deep learning development**, running on top of TensorFlow.

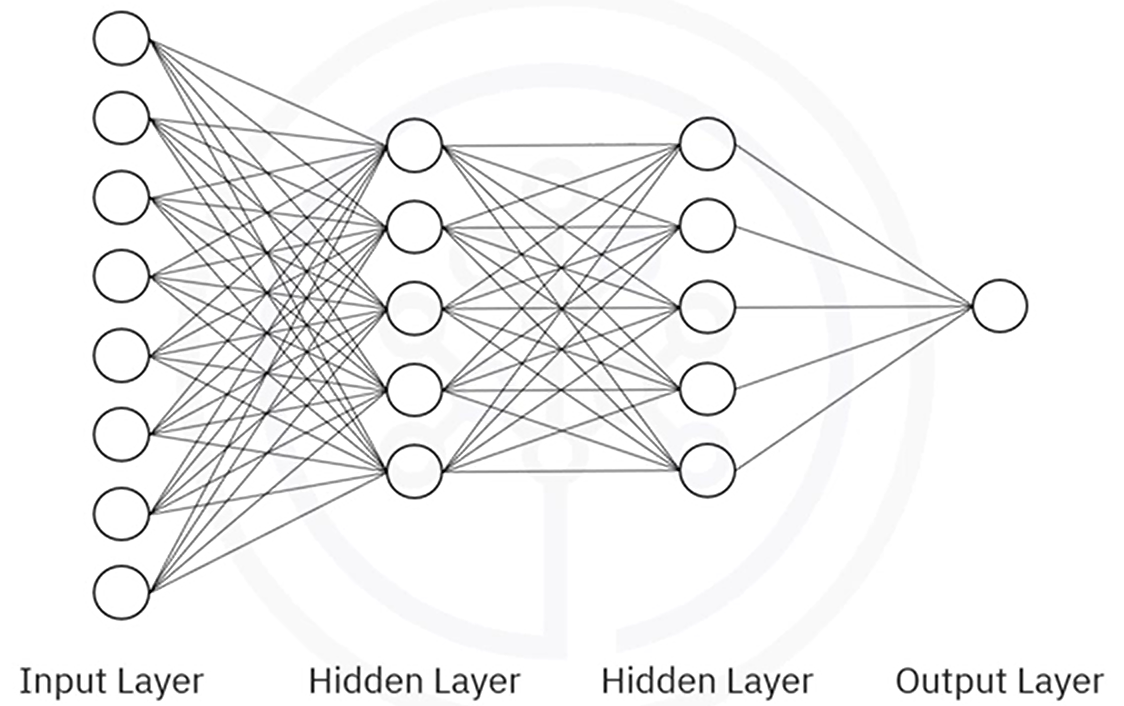
✅ Beginners are encouraged to start with **Keras** for its ease of use, while advanced users may leverage **PyTorch or TensorFlow** for fine-grained control.

## 📌 Regression Models with Keras

This section introduces how to build and train a **regression model** using the **Keras** library. The goal is to predict a continuous value based on a dataset.

Keras allows you to construct and train deep learning models with **minimal code** and is ideal for quickly prototyping regression tasks using dense, feedforward neural networks.

### 🔹 Building the Network Architecture



The model will consist of:

* **Input layer**: receives features.
* **Hidden layers**: apply transformations to extract patterns.
* **Output layer**: outputs a single continuous value.

✅ In hidden layers, it is standard practice to use **ReLU activation functions**.

✅ The output layer typically has **no activation function**, since it’s predicting a continuous value.

This type of network is called a **dense (fully connected) network**, where each neuron in one layer is connected to every neuron in the next.

While deeper or wider networks (e.g., with 50–100 neurons per layer) are often used in practice, smaller networks can also illustrate core concepts.

### 🔹 Using Keras to Define the Model

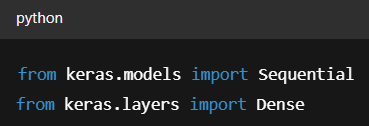
Keras provides two ways to define models:

1. **Sequential API**: Used when layers are stacked one after the other (most common).
2. **Functional API**: Used for more advanced or non-linear architectures.

In most regression use cases; the **Sequential model** is appropriate.

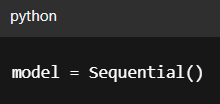
**Step-by-step:**

1. **Import required modules:**

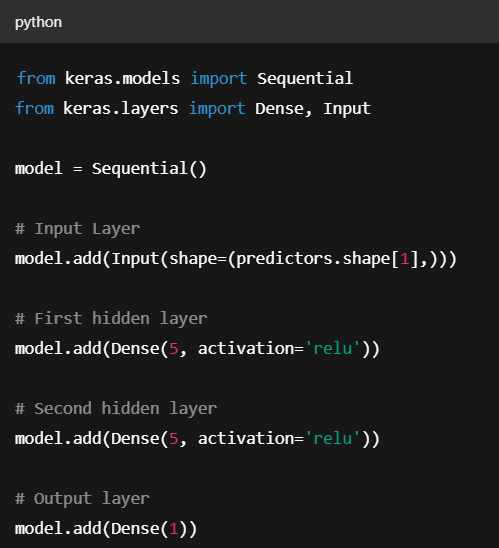


1. **Initialize the model:**

Use the Sequential API to stack layers.

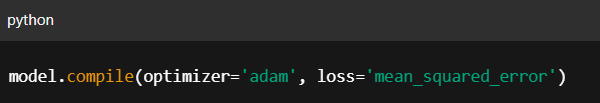
****

1. **Add Layers:**



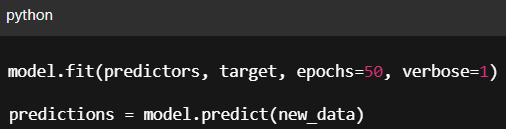
* + The **first layer** specifies the number of input features via input\_shape parameter.
  + Each **hidden layer** uses **ReLU** to introduce non-linearity and improve learning.
  + The **final output layer** has one neuron with no activation (for raw value output).

1. **Compiling the Model:**

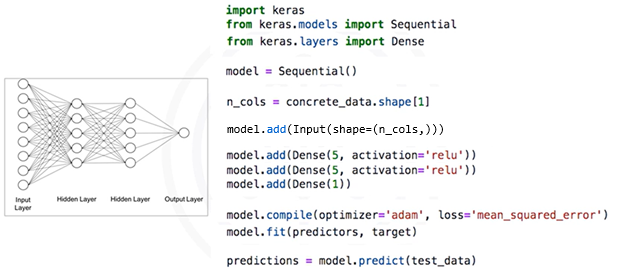
****

* + The **Adam optimizer** is commonly used in deep learning for its adaptive learning rate and efficiency , you don’t need to manually set a learning rate.
  + **Mean Squared Error (MSE)** is the standard loss function for regression as it penalizes large errors more than small ones.

1. **Training and Predicting:**



**All Together:**



### 🔹 Takeaways

✅ Keras simplifies the process of building and training regression models using a **layered API** and sensible defaults.

✅ Input data must be separated into **predictors** and a **target**, with proper formatting for compatibility with neural network inputs.

✅ Use **Dense layers** for fully connected architectures, **ReLU** activations in hidden layers, and **no activation** in the output layer for continuous predictions.

✅ Compile the model with the **Adam optimizer** and **mean squared error loss** to effectively train on regression problems.

✅ The entire workflow—from data preparation to model training and prediction—can be implemented clearly and efficiently, allowing for fast prototyping and model deployment.

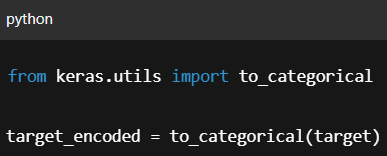
## 📌 Classification Models with Keras

Classification models are built in Keras much like regression models: using the **Sequential API**, stacking Dense layers, and compiling with an optimizer and loss function.

However, there are **key differences** in how the data is prepared, how the output layer is configured, and what loss function and metrics are used.

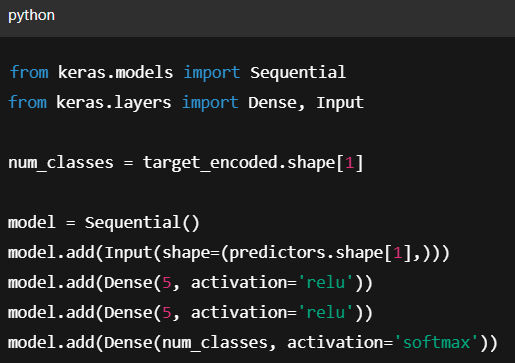
**Step-by-step:**

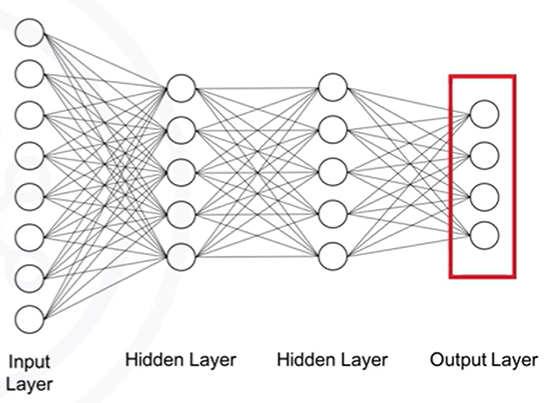
1. **Preparing data:**



* + Just like in regression, we begin by splitting the dataset into **predictors** and **target**, but for classification, the target column contains **class labels**, which must be **one-hot encoded** before training.
  + Each row of the output represents a class with a 1 at the correct index and 0s elsewhere, If there are 4 classes, each target row becomes a vector like [0, 0, 1, 0].
  + This transformation is required to match the **output layer’s softmax structure** and for the loss function to compute properly.

1. **Network Architecture:**

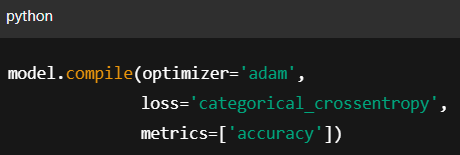
****

* ****The core network structure (input → hidden layers → output) remains the same as in regression, but there are **two important architectural differences**:

1. **The output layer must have as many neurons as there are classes.**
2. **The activation function of the output layer must be softmax.**

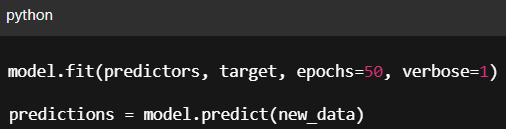
This allows the model to produce a **probability distribution** over the possible classes.

1. **Compiling the Model:**

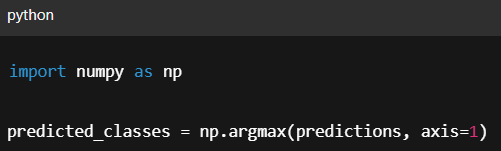
****

* + For classification, we switch the loss function to be ‘**Categorical crossentropy**’, this function is appropriate when the targets are one-hot encoded and the model outputs class probabilities.

1. **Compiling the Model:**



* + Just like with regression, each epoch passes the full dataset through the network, updates weights via backpropagation, and improves the model’s ability to predict the correct class.
  + Once the model is trained, we can make predictions using ***.predict()*.** This returns **class probabilities**, for example, if there are 4 classes, a prediction might look like

-> **[0.01, 0.92, 0.04, 0.03]**

To get the predicted class label:

### 🔹 Takeaways

✅ The overall structure for classification and regression models in Keras is nearly identical, but classification requires:

* Transforming categorical targets using **to\_categorical()**.
* Using a **softmax output layer**.
* Setting **categorical crossentropy** as the loss function.
* Evaluating using **accuracy** rather than a numerical error metric.

✅ Keras handles the internals of class probability calculation and training, so most of the additional steps are related to **data preparation and model configuration**.

📌 💡 ✅ ❌ 🚫 ⚠️ **🔍** ℹ️ **🧠** ⚙️ 🔧 **🧰** 🛠 🧩